Catatan Kuliah Sistem Terdistribusi

Ruddy J. Suhatril, SKom

27 Maret 2004

Daftar Isi

Daftar Gambar

Bab 1

Pendahuluan

1.1 Apakah yang dimaksud dengan Sistem Terdistribusi ?

Sistem Terdistribusi adalah Sekumpulan komputer otonom yang terhubung ke suatu jaringan, dimana bagi pengguna sistem terlihat sebagai satu komputer. Maksud komputer otonomi adalah walaupun komputer tidak terhubung ke jaringan, komputer tersebut tetap data berjalan.

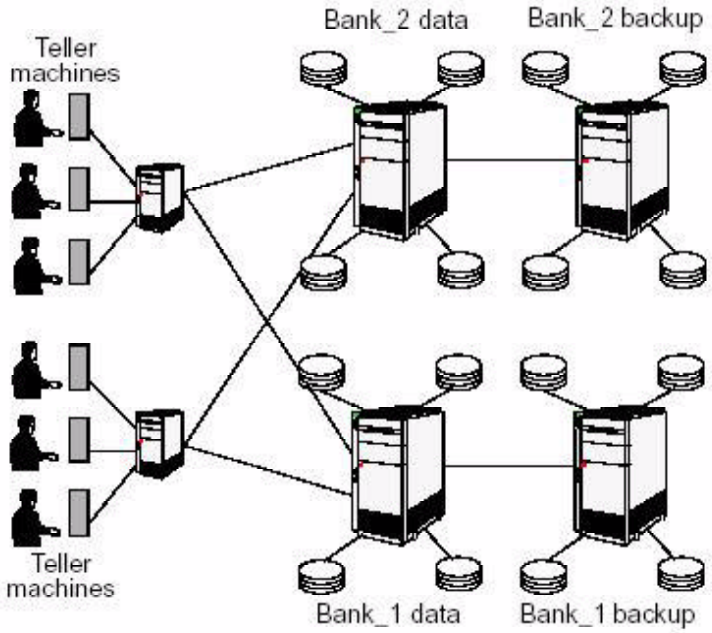
Dengan menjalankan sistem terdistribusi, komputer dapat melakukan :

- Koordinasi Akti...tas
- Berbagi sumber daya : hardware, software dan data

Dengan de...nisi tersebut diatas maka internet sesungguhnya bukanlah suatu sistem terdistribusi, melainkan infrastruktur dimana sistem terdistribusi dapat di aplikasikan pada jaringan tersebut.

1.2 Contoh Sistem Terdistribusi

- Sistem Telepon
  - ISDN, PSTN
- Manajemen Jaringan
  - Adminstrasi sesumber jaringan



Gambar~1.1: Contoh sistem terdistribusi, Automatic Banking (teller machine) System

- Network File System (NFS)
  - Arsitektur untuk mengakses sistem ...le melalui jaringan
- WWW
  - Arsitektur client/server yang diterapkan di atas infrastruktur internet
  - Shared Resource (melalui URL)
- dll...

1.3 Keuntungan dan Permasalahan Sistem Terditribusi

1.3.1 Keuntungan Sistem Terdistribusi

Keuntungan yang didapatkan dalam menerapkan sistem terdistribusi, antara lain :

- Performance

  Kumpulan dari beberapa prosesor akan memberikan kinerja yang lebih baik dari pada komputer yang terpusat. Begitu juga kalau dilihat dari sisi biaya.

- Distribution

- Reliability (Fault tolerance)

  apabila salah satu komponen terjadi kerusakan, system tetap dapat berjalan

- Incremental Growth

  Mudah dalam melakukan penambahan komputer/komponen

- Sharing Data/Resources

  Berbagi data adalah salah satu hal yang pokok pada kebanyakan aplikasi.

1.3.2 Permasalahan dalam Sistem Terdistribusi

Kelemahan pada sistem terdistribusi adalah :

- Kesulitan dalam membangun perangkat lunak .

  Kesulitan yang akan dihadapi antara lain : bahasa pemrogramman yang harus dipakai, sistem operasi dll.

- Masalah Jaringan

  Karena sistem terdistribusi di implementasikan dalam jaringan komputer, maka isu2 yang berkaitan dengan jaringan komputer akan menjadi pertimbangan utama dalam merancang dan mengimplementasikan sistem.

- Masalah Keamanan

  Karena pada sistem terdistribusi berbagi data/sumber daya merupakan hal yang mutlak maka muncul masalah2 yang berkaitan dengan keamanan data dll.

1.4 Karakteristik Sistem Terdistribusi

Ada beberapa hal yang harus diperhatikan dalam membangun sistem terdistribusi, yaitu :

- Transparency (Kejelasan)
- Communication (Komunikasi)
- Performance & Scalability (Kinerja dan Ruang Lingkup)
- Heterogeneity (Keanekaragaman)
- Opennes (Keterbukaan)
- Reliability & Fault Tolerancy (Kehandalan dan Toleransi Kegagalan)
- Security (Kemanan)

1.4.1 Transparency

Access transparency

Sumber daya lokal dan remote di akses dengan menggunakan operasi yang sama.

Location transparency

Pengguna sistem tidak tahu mengetahui keberadaan hardware dan software (CPU,...le dan data).

Migration (Mobility) transparency

Sumber daya (baik berupa Hardware dan/atau software) dapat bebas berpindah tanpa mengubah sistem penamaan.

Replication transparency

Sistem bebas untuk menambah ...le atau sumber daya tanpa diketahui oleh user (dalam rangkan meningkatkan kinerja)

Concurency transparency

User tidak akan mengetahui keberadaan user lain dalam sistem, walaupun user tersebut menggunakan sumber daya yang sama.

Failure transparency

Aplikasi harus dapat menyelesaikan proses nya walaupun terdapat kegagalan pada beberapa pada komponen sistem.

Performance transparency

Beban kerja yang bervariasi tidak akan menyebabkan turunnya kinerja sistem, hal ini dapat di capai dengan melakukan automatisasi kon...gurasi terhadap perubahan beban.

1.4.2 Communication

Komponen2 pada sistem terdistribusi harus melakukan komunikasi dalam suatu urutan. Sebagai berikut :

- Infrastruktur jaringan (interkoneksi dan software jaringan)
- Metode dan Model komunikasi yang cocok

  Metode komunikasi :

  - Send
  - Receive
  - Remote Procedure Call

  Model Komunikasi

  - client - server communication : pertukaran pesan antara dua proses : dimana satu proses (client) menggunakan / meminta layanan pada server dan server menyediakan hasil dari proses tersebut.

- group mulitcast : target dari pesan yang dikirimkan adalah gabungan dari proses, yang berasal dari suatu grup.

1.4.3 Performance and Scalability

Ada beberapa faktor yang mempengaruhi kinerja (performance) dari pada sistem terdistribusi :

- Kinerja dari pada personal workstations
- Kecepatan infrastruktur komunikasi
- Fleksibilitas dalam membagi beban kerja : contoh, apabila terdapat prosesor (workstation) yang idle maka dapat di alokasikan secara otomatis untuk mengerjakan tugas2 user.

Scalability

Sistem tetap harus memperhatikan efesiensi walaupun terdapat penambahan secara signi…kan user atau sumber daya yang terhubung :

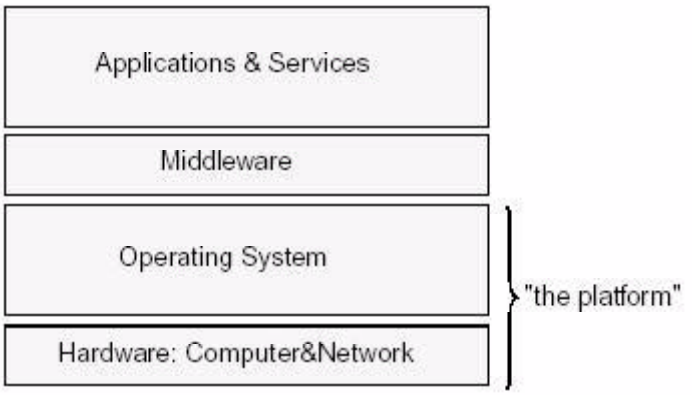
- Cost (biaya) penambahan sumber daya (resources) harus reasonable.
- Penurunan kinerja (performance) diakibatkan oleh penambahan user atau sumber daya harus terkontrol.

1.4.4 Heterogeneity

Aplikasi yang terdistribusi biasa berjalan dalam keberagaman :

- Hardware : mainframes, workstations, PC's, server dll.
- Software : UNIX, MS Windows, IMB OS/2, LINUX dll.
- Devices : teller machine, robot, sistem manufacturing dll.
- Network dan Protocol : Ethernet, FDDI, ATM, TCP/IP dll

Melihat keaneka ragaman di atas maka salah satu solusi yang bisa di terapkan adalah Middleware : berfungsi sebagai jembatan untuk komunikasi dan proses.



Gambar~1.2: Arsitektur sofware pada sistem terdistribusi

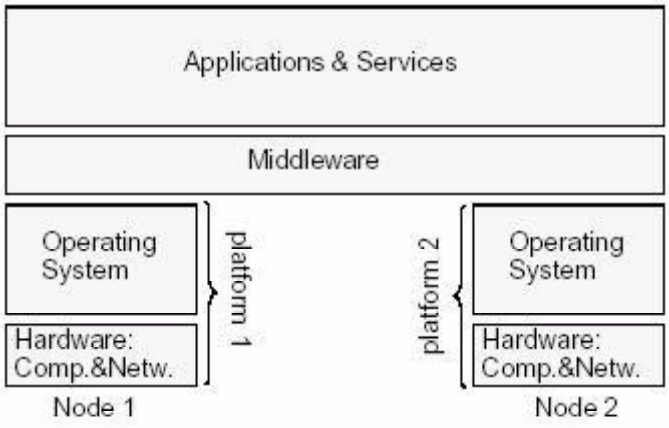
1.4.5 Opennes

Salah satu hal terpenting yang harus dimiliki oleh sistem terdistribusi adalah opennes (keterbukaan) dan ‡exibility (‡eksibilitas) :

- Setiap layanan (services) harus dapat di akses oleh semua user.
- Mudah dalam implementasi, install dan debug services;
- User dapat membuat dan menginstall service yang telah dibuat oleh si user tersebut.

Aspek kunci pada opennes :

- Interface dan Protocol yang standard (seperti protokol komunikasi di internet)
- Support terhadap keanekaragaman. (dengan membuat midleware seperti CORBA)



Gambar~1.3: Sistem Terdistribusi pada dua titik

1.4.6 Reliability dan Fault Tolerance

Salah satu tujuan dalam membangun sistem terdistribusi adalah memunkinkan untuk melakukan improvisasi terhadap kehandalan sistem.

Availability : kalau mesin mati (down), sistem tetap harus berjalan dengan jumlah layananan yang tersisa.

- Dalam sistem terdistribusi componen yang sangat vital (critical resources) berjumlah se minimal mungkin. Yang dimaksud dengan critical resources adalah komponen yang harus ada untuk menjalankan sistem terdistribusi.
- Masing - masing Software dan Hardware harus di replikasi : kalau terjadi kegagalan / error maka yang lain akan menangani.

Data dalam sistem tidak boleh hilang, copy dari …le tersebut disimpan pada secara redundan pada server lain, tapi tetap harus dijaga konsistensi datanya.

Fault Tolerance : Sistem harus bisa mendeteksi kegagalan dan melakukan tindakan dengan dasar sebagai berikut :

- Mask the fault (menutupi kegagalan) : tugas harus dapat dilanjutkan dengan menurunkan kinerja tapi tanpa terjadi kehilangan data atau informasi.

- Fail Gracefully : membuat suatu antisipasi terhadap suatu kegagalan ke suatu prosedur yang telah di rencanakan dan memungkinkan untuk menghentikan proses dalam waktu yang singkat tanpa menghilangkan informasi atau data.

1.4.7 Security

- Con...dentiality :

  keamanan terhadap data yang di akses oleh user yang tidak di perbolehkan (unauthorizes user)

- Integrty:

  keamanan terhadap kelengkapan dan autentikasi data.

- Availability

  Menjaga agar resource dapat selalu di akses.

Sistem terdistribusi harus memperbolehkan komunikasi antara program/user/resources pada computer yang berbeda, maka resiko keamanan akan muncul apabila memberlakukan free access. Dan ada hal lain juga yang harus dijamin dalam sistem terdistribusi, yaitu : penggunaan rerources yang tepat oleh user yang berlainan.

1.5 Model dalam Sistem Terdistribusi

Model dalam sistem terdistribusi :

- Model Arsitektur (Architectural Models)
- Model Interaksi (Interaction Models)
- Model Kegagalan (Failure Models)

Resources dalam sistem terdistribusi dipakai secara bersama oleh users. Biasa nya di bungkus (encapsulated) dalam suatu komputer dan dapat di akses oleh komputer lain dengan komunikasi.

Setiap resource di atur oleh program yang disebut dengan resource manager. Resource manager memberikan kemungkinan komunikasi interface antar resource.

Resource Managers dapat digeneralisasi sebagai proses, kalau sistem di design dengan sudut pandang object (Object Oriented), resource dibungkus dalam suatu objek.

1.5.1 Architectural Models

Bagaimana cara kerja sistem terdisribusi antara komponen - komponen sistem dan bagaimana komponen tersebu berada pada sistem terdistribusi :

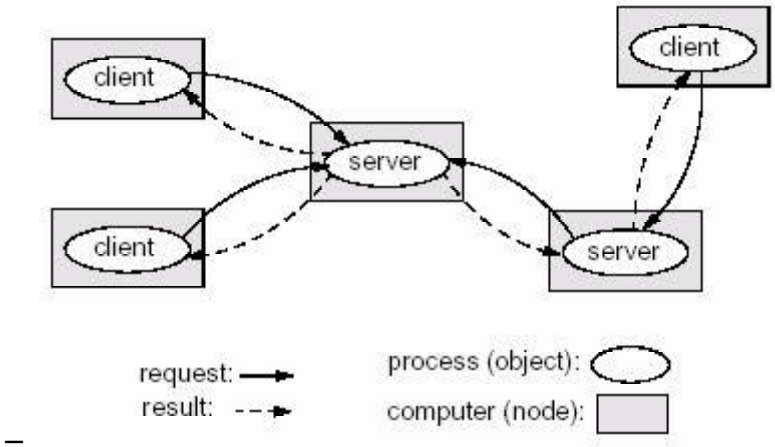
- Client - Server Model
- Proxy Server
- Peer processes (peer to peer)

Client - Server Model

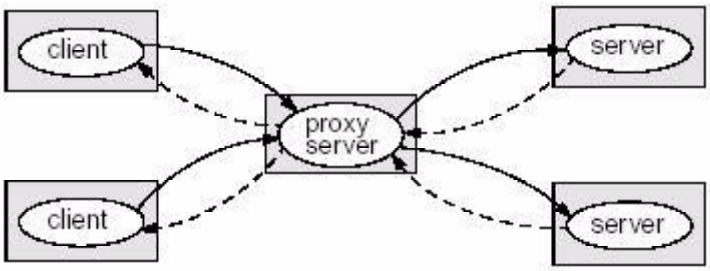
Sistem yang terdiri dari kumpulan2 proses disebut dengan server, dan memberikan layanan kepada user yang disebut dengan client.

Model client-server biasanya berbasiskan protokol request/reply. Contoh implementasi nya, atara lain: RPC (Remote Procedure Calling) dan RMI (Remote Method Invocation) :

- client mengirimkan request berupa pesan ke server untuk mengakses suatu service.
- server menerima pesan tersebut dan mengeksekusi request client dan mereply hasil ke client



Gambar~1.4: Model arsitektur client - server



Gambar~1.5: Model Proxy Server

Proxy Server

Proxy server menyediakan hasil copy (replikasi) dari resource yang di atur oleh server lain

Biasa nya proxy server di pakai untuk menyimpan hasil copy web resources. Ketika client melakukan request ke server, hal yang pertama dilakukan adalah memeriksa proxy server apakah yang dimita oleh client terdapat pada proxy server.

Proxy server dapat diletakkan pada setiap client atau dapat di pakai bersama oleh beberapa client.

Tujuannya adalah meningkatkan performance dan availibity dengan mencegah frekwensi akses ke server.

Peer Process

Semua proses (object) mempunyai peran yang sama.

- Proses berinteraksi tanpa ada nya perbedaan antara client dan server.
- Pola komunikasi yang digunakan berdasarkan aplikasi yang digunakan.
- Merupakan model yang paling general dan ‡eksible.

1.5.2 Interaction Models

Untuk interaksi nya sistem terdistribusi dibagi menjadi dua bagian :

- Synchrounous distributed system
- Asynchronous distributed system

Synchronous Distributed System

Batas atas dan batas bawah waktu pengeksekusian dapat di set.

- Pesan yang dikirim di terima dalam waktu yang sudah di tentukan
- Fluktuasi ukuran antara waktu local berada dalam suatu batasan.

Beberapa hal yang penting untuk di perhatikan :

- Dalam synchronous distributed system terdapat satu waktu global.
- Hanya synchronous distributed system dapat memprediksi perilaku (waktu).
- Dalam synchornous distributed system dimungkinkan dan aman untuk menggunakan mekanisme timeout dalam mendekteksi error atau kegagalan dalam proses atau komunikasi.

Asynchronous Distributed System

Banyak sistem terdistribusi yang menggunakan model interaksi ini (termasuk Internet)

- Tidak ada batasan dalam waktu pengkeksekusian.
- Tidak ada batasan dalam delay transmission (penundaan pengiriman)
- Tidak ada batasan terhadap ‡uktuasi waktu local.

Asynchronous system secara parktek lebih banyak digunakan.

1.5.3 Failure Models

Kegagalan apa saja yang dapat terjadi dan bagaimana efek yang ditimbulkan ?

- Omission Faluires
- Arbitary Failures
- Timing Failures

Kegagalan dapat terjadi pada proses atau kanal komunikasi. Dan penyebabnya bisa berasal dari hardware ataupun software.

Model Kegagalan (Failure Models) dibutuhkan dalam membangun suatu sistem dengan prediksi terhadap kagagalan2 yang mungkin terjadi.

Ommision Failures

Yang dimaksud dengan Ommision Failures adalah ketika prosesor dan kanal komunikasi mengalami kegagalan untuk melakukan hal yang seharusnya dilakukan. Dikatakan tidak mempunyai ommision failures apabila :

- Terjadi keterlambatan (delayed) tetapi akhirnya tetap tereksekusi.
- Sebuah aksi di eksekusi walaupun terdapat kesalahan pada hasil.

Dengan synchronous system, ommision failures dapat dideteksi dengan timeouts. Kalau kita yakin bahwa pesan yang dikirim sampai, timeout akan mengindikasikan bahwa proses pengiriman rusak, seperti fail-stop behavior pada sistem.

Arbitary Failures

Ini adalah kegagalan yang paling buruk dalam sistem. Tahapan proses atau komunikasi diabaikan atau yang tidak diharapkan terjadi dieksekusi. Sehingga hasil yang diharapkan tidak terjadi atau megeluarkan hasil yang salah.

Timing Failures

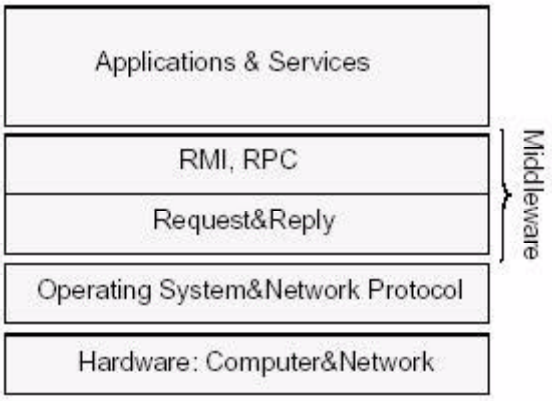
Timing Failures dapat terjadi pada synchronous system, dimana batas waktu di atur untuk eksekusi proses, komunikasi dan ‡uktuasi waktu. Timing Failures terjadi apabila waktu yang telah ditentukan terlampaui.

Bab 2

Komunikasi

2.1 Sistem Komunikasi

Pada bab ini akan dibahas bagaimana komunikasi antara object2 dalam sistem terdistribusi, khusus nya dengan menggunakan RMI (Remod Method Invokation) dan RPC (Remote Procedure Call). RMI dan RPC berbasiskan metode request dan reply.

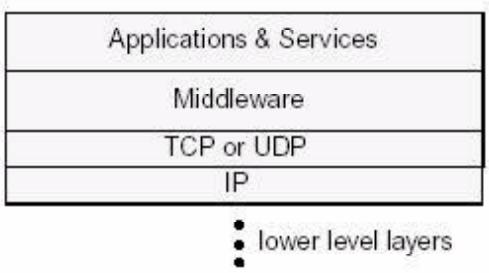


Gambar~2.1: Model komunikasi dan implementasi layer pada sistem terdistribusi

Request dan repy diimplementasikan pada protokol jaringan.

2.2 Network Protocol

Middleware dan aplikasi terdistribusi di implementasikan diatas protokol network.Protocol diimplementasikan dalam beberapa lapisan (layer).



Layer protocol pada Internet

2.2.1 TCP dan UDP

TCP

TCP (Transport Control Protocol) dan UDP (User Datagram Protocol) adalah protokol transport yang berada di atas lapisan Internet Protocol (IP).

TCP adalah protocol yang handal, TCP dapat memastikan data yang dikirimkan sampai ke tujuan begitu juga sebaliknya.

TCP menambahkan beberapa prosedur diatas layer internet protocol untuk memastikan reliabilitas transport data :

- Sequencing

  Pada setiap transmisi data (paket) diberi nomor urut. Sehingga pada titik tujuan tidak ada segmen yang diterima sampai semua segmen pada urutan bawah belum di terima.

- Flow Control

  Pengirim tidak akan membanjiri penerima.Karena pengiriman didasarkan pada periode acknowledgment yang di terima oleh pengirim yang berasal dari penerima.

- Retrasnmission dan duplicate handling

Apabila segmen tidak mendapatkan acknowledge dari penerima sampai waktu timeout yang ditentukan terlampaui maka pengirim akan mengirim ulang. Berdasarkan nomor urut penerima data dapat mendeteksi dan menolak kalau terjadi duplikasi.

- Bu¤ering

  Bu¤ering digunakan untuk menyeimbangkan antara pengirim dan penerima. Kalau bu¤er pada penerima penuh, maka segmen yang datang akan putus, sehingga menyebabkan tidak ada acknowledge ke pengirim dan pengirim akan melakukan transmot ulang.

- Checksum

  Setiap segment membawa checksum. Apabila checksum segmen yang di terima tidak sesuai maka paket data tersebut akan di drop (dan kemudian akan di transmit ulang)

UDP

UDP tidak memberikan garansi seperti halnya yang di berikan oleh TCP.

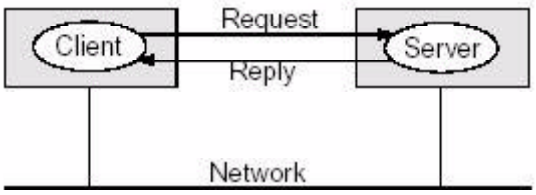
- UDP tidak memberikan garansi terhadap pengiriman data

  Pada Internet Protocol paket data dapat drop karena suatu hal contohnya jaringan yang rusak, UDP tidak mempunyai mekanisme untuk menanggulangi hal tersebut.

- Kalau ingin menggunakan UDP sebagai protocol pengiriman yang handal, maka mekanisme kehandalan yang diinginkan di lakukan pada layer aplikasi.

2.2.2 Komunikasi Request - Reply

Komunikasi antara proses dan objek pada sistem terdistribusi dilakukan melalui message passing.

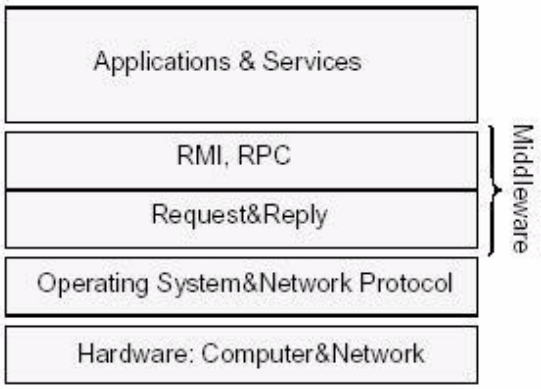


Client melakukan :
1. Mengirim (request) pesan ke server
2. Menerima hasil (reply dari server)

Server melakukan :
1. Penerimaan pesan (request) dari client
2. Mengeksekusi permintaan dari client
3. Mengirim hasil (reply) ke client.

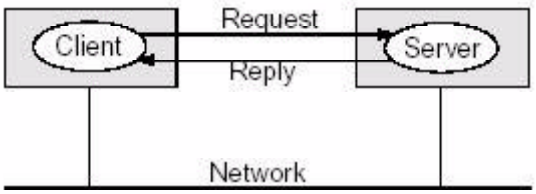
2.3 RPC dan RMI

Tujuan dari RPC dan RMI dibuat bagi programmer, agar computer yang terdistribusi terlihat seperti computer yang terpusat. Dan berguna untuk melihat sistem terdistribusi dari sisi pemrogramman.



RPC dan RMI berada pada Middleware

2.3.1 RMI (Remote Method Invocation)



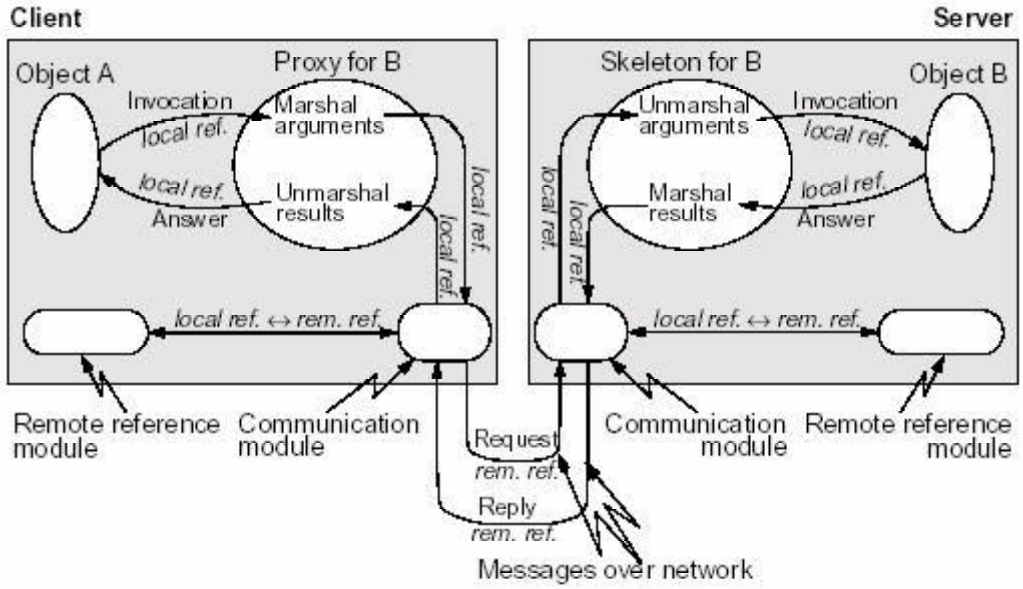
Berikut ilustrasi yang terjadi pada metode RMI :
Programmer pada client menulis :

server_id.service(values_to_server,result_arguments);

Pada sisi server mempunyai fungsi sebagai berikut :

public service(in type1 arg from client; out type2 arg to_client)
{————};

Programmer pada client tidak mengetahui bahwa reply message yang didapatkan berasal dari server yang dikirim melalui jaringan.



Gambar~2.2: Ilustrasi implementasi RMI

Komponen2 dalam RMI (gambar 2.2):

- Object A (client) : meminta layanan
- Object B (server) : menghantarkan layanan
- Proxy for B
  - Ketika object A mempunyai remote reference ke object B, maka akan timbul objek Proxy B pada host object A. Proxy terbuat ketika remote object reference digunakan pertama kali

- Proxy adalah perwakilan objek yang berada pada remote, dengan kata lain ketika terjadi invokasi dari A ke B ditangani seolah - olah hanya mengakses Proxy B.
- Ketika invokasi terjadi proxy menggunakan metode marshals untuk membungkus pesan yang akan dikirim ke server. Dan setelah menerima hasil dari server proxy menggunakan metode unmarshal (membuka bungkus) untuk kemudian diteruskan ke client (Object A)

² Skeleton for object B

- Pada sisi server, terdapat object kerangka (skeleton) yang berhubungan ke class, kalau object pada class tersebut dapat diakses oleh RMI.
- Skeleton menerima pesan kemudian melakukan unmarshal dan meneruskan ke method object yang dituju. Dan kemudian menunggu hasil dari object B dan kemudian membungkus hasil (unmarshal) dan kemudian dikirimkan ke client (Objec A).
- Ada bagian dari skeleton B yang disebut dengan dispatcher. dispatcher menerima request dari communication module, dan kemudian mengidenti…kasi invokasi dan mengarahkan permintaan ke corresponding method (method pada skeleton yang berfungsi untuk berkomunikasi dengan object).

² Communication Modul (Modul Komunikasi)

-Communication modul pada client atau server bertanggung jawab dalam pertukaran pesan yang dilakukan melalui metode request dan reply.

² Remote Reference Module

- Bagian ini bertugas untuk menterjemahkan antara referensi objek lokal dan remote. Proses berkomunikasi antara mereka disimpan dalam remote object table.

Yang mengenerate class untuk proxy dan skeleton adalah middleware.
contoh : CORBA, Java RMI

Object A dan object B dipunyai oleh aplikasi (berada pada Application Layer)

Remote Reference Modul dan Communication modul dimiliki oleh middleware.

Proxy B dan Sekeleton B berada antara middleware dan aplikasi yang di generate oleh middleware.

Langkah2 proses dengan RMI :

- Urutan pemanggilan pada object client mengaktifkan method pada proxy yang akan berhubungan dengan invoked method (method yang ter-invokasi) pada object B.
- Kemudian method yang ada pada proxy melakukan pembungkusan argumen menjadi suatu pesan (marshalling) dan meneruskan ke modul komunikasi.
- Berdasarkan pada remote reference yang didapat dari remote reference modul,modul komunikasi memulai request dan reply protocol melalui network.
- Modul komunikasi pada server menerima request dari client. Kemudian berdasarkan referensi lokal yang diterima dari remote reference modul maka akan mengaktifkan method untuk berkomunikasi dengan object pada skeleton B (corresponding method).
- Method pada skeleton meng-ekstrak (unmarshalling) argumen pada pesan yang di terima dan mengaktifkan corresponding method (method yang berfungsi untuk melakukan komunikasi) pada object B (server).
- Setelah menerima hasil dari object B, method dari skeleton akan membungkus hasil tersebut dalam sebuah pesan (marshalling) dan meneruskan pesan yang sudah dibungkus ke modul komunikasi.
- Modul komunikasi mengrimkan pesan tersebut ke client melalui jaringan.
- Modul komunikasi pada client menerima hasil (reply) dari server dan meneruskan ke corresponding method pada proxy.

- Kemudian proxy meng-ektrak hasil (unmarshalling) dan meneruskan ke object A (client).

Contoh RMI dengan menggunakan Java RMI :

Server object akan mencetak "Hello Ruddy" ke layar & mengembalikan pesan ke klien

Pada sisi server :

- Server Method

```
import java.rmi.*;
public interface SimpleInterface extends Remote {
 String printMessage(String name) throws RemoteException;
}
```

- Server Object

```
import java.rmi.*;
import java.rmi.server.*;
public class SimpleServer extends UnicastRemoteObject implements SimpleInterfac
 public SimpleServer() throws RemoteException { super(); }
 public String printMessage(String name) throws RemoteException {
 System.out.println(name);
 return(Hello  + name);
 }

public static void main(String args[]) {
 System.setSecurityManager(new RMISecurityManager());
 try {
 SimpleServer newServer = new SimpleServer();
 System.out.println(SimpleServer attempting to
 bind to the registry);
 Naming.rebind(//ruddy.info:30010/SimpleServer,
 newServer);
 System.out.println(SimpleServer bound in the registry);
 } catch(Exception e) {
```

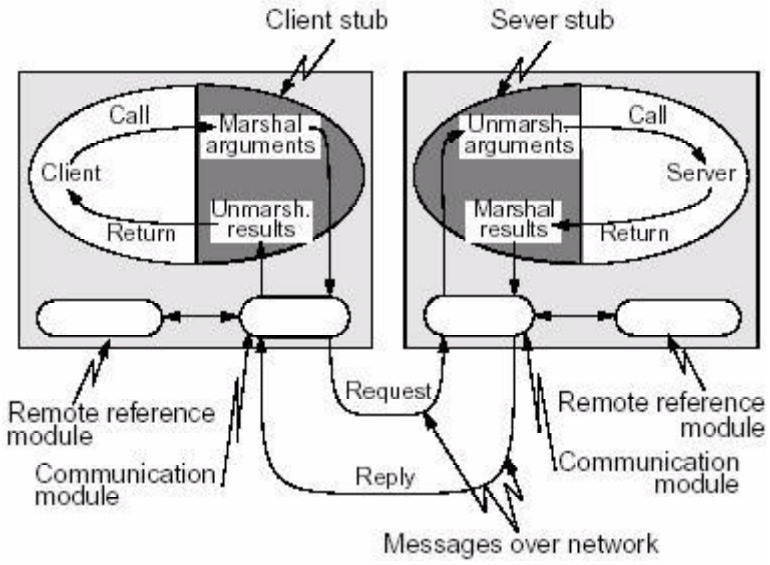
```
 System.out.println(SimpleServer error:  + e.getMessage());
 e.printStackTrace();

 }
 }
}
```

Pada sisi client :

```
import java.rmi.*;
public class SimpleClient {
 private static SImpleInterface server = null;
 public static void main(String args[]) {
 try {
 server = (SimpleInterface)
 Naming.lookup(//ruddy.info:30010/SimpleServer);
 System.out.println(server.printMessage(Ruddy));
 } catch(Exception e) {
 System.out.println(SimpleClient error: + e.getMessage());
 e.printStackTrace();
 }
 }
}
```

2.3.2 RPC (Remote Procedure Call)

Proses nya kurang lebih sama dengan RMI. Kalau RMI kita mengenal Proxy dan Skeleton, pada RPC dikenal dengan Stub (Client Stub dan Server Stub).



Gambar~2.3: Ilustrasi implementasi RPC

Remote Reference Modul dan Communication Modul berada pada tatanan sistem operasi.

Bab 3

Proses

3.1 Konsep Proses

Jika kita berdiskusi mengenai sistem operasi, maka akan timbul sebuah pertanyaan yaitu mengenai istilah apa yang tepat untuk menyebut semua kegiatan yang dilakukan oleh CPU. Sistem batch mengeksekusi jobs sebagaimana suatu sistem time-share menggunakan program pengguna (user programs) atau tasks. Bahkan pada sistem dengan pengguna tunggal pun, seperti pada Microsoft Windows dan Macintosh OS, seorang pengguna mampu menjalankan beberapa program pada saat yang sama, contohnya Word Processor, Web Browser, dan paket e-mail. Bahkan jika pengguna hanya dapat menjalankan satu program pada satu waktu, sistem operasi perlu untuk mendukung aktivitas program internalnya sendiri, seperti managemen memori. Dalam banyak hal, seluruh aktivitas ini adalah serupa, maka kita menyebut seluruh program itu proses-proses.

Istilah job dan proses digunakan hampir dapat dipertukarkan pada tulisan ini. Walau kami sendiri lebih menyukai istilah proses, banyak teori dan terminologi sistem operasi dikembangkan selama suatu waktu ketika aktivitas utama sistem operasi adalah job processing. Akan membingungkan jika kita menghindari penggunaan istilah yang telah diterima oleh masyarakat yang memasukkan kata job hanya karena proses memiliki istilah job sebagai pengganti atau pendahulunya.

3.1.1 De…nisi Proses

Secara tidak langsung, proses merupakan program yang sedang dieksekusi. Menurut Silberschatz, suatu proses adalah lebih dari sebuah kode program, yang terkadang disebut text section. Proses juga mencakup program counter, yaitu sebuah stack untuk menyimpan alamat dari instruksi yang akan dieksekusi selanjutnya dan register. Sebuah proses pada umumnya juga memiliki sebuah stack yang berisikan data-data yang dibutuhkan selama proses dieksekusi seperti parameter metoda, alamat return dan variabel lokal, dan sebuah data section yang menyimpan variabel global.

Sama halnya dengan Silberschatz, Tanenbaum juga berpendapat bahwa proses adalah sebuah program yang dieksekusi yang mencakup program counter, register, dan variabel di dalamnya.

Kami tekankan bahwa program itu sendiri bukanlah sebuah proses; suatu program adalah satu entitas pasif; seperti isi dari sebuah berkas yang disimpan didalam disket. Sedangkan sebuah proses dalam suatu entitas aktif, dengan sebuah program counter yang menyimpan alamat instruksi selanjut yang akan dieksekusi dan seperangkat sumber daya (resource) yang dibutuhkan agar sebuah proses dapat dieksekusi.

Untuk mempermudah kita membedakan program dengan proses, kita akan menggunakan analogi yang diberikan oleh Tanenbaum. Misalnya ada seorang tukang kue yang ingin membuat kue ulang tahun untuk anaknya. Tukang kue tersebut memiliki resep kue ulang tahun dan bahan-bahan yang dibutuhkan untuk membuat kue ulang tahun di dapurnya seperti: tepung terigu, telur, gula, bubuk vanila dan bahan-bahan lainnya. Dalam analogi ini, resep kue ulang tahun adalah sebuah program, si tukang kue tersebut adala prosesor (CPU), dan bahan-bahan untuk membuat kue tersebut adalah data input. Sedangkan proses-nya adalah kegiatan sang tukang kue untuk membaca resep, mengolah bahan, dan memanggang kue tersebut.

Walau dua proses dapat dihubungkan dengan program yang sama, program tersebut dianggap dua urutan eksekusi yang berbeda. Sebagai contoh, beberapa pengguna dapat menjalankan salinan yang berbeda pada mail program, atau pengguna yang sama dapat meminta banyak salinan dari program editor. Tiap-tiap proses ini adakah proses yang berbeda dan walau bagian text-section adalah sama, data section-nya bervariasi. Adalah umum untuk memiliki proses yang menghasilkan banyak proses begitu ia bekerja.

3.1.2 Status Proses

Bila sebuah proses dieksekusi, maka statusnya akan berubah-ubah. Status dari sebuah proses mencerminkan aktivitas atau keadaan dari proses itu sendiri. Berikut ini adalah status-status yang mungkin dimiliki sebuah proses menurut Tanenbaum:

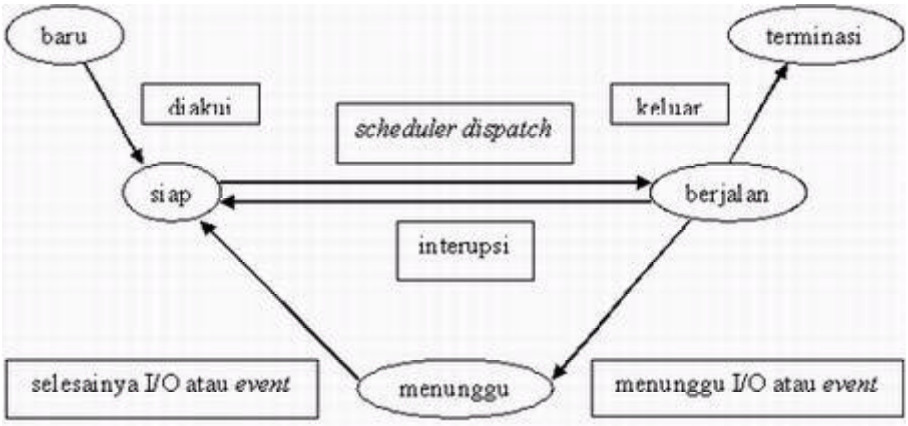
- Running: pada saat menggunakan CPU pada suatu waktu.
- Ready: proses diberhentikan sementara karena menunggu proses lain untuk dieksekusi.
- Blocked: tidak dijalankan sampai event dari luar, yang berhubungan dengan proses tersebut terjadi.

Sedangkan menurut Silberschatz, terdapat lima macam jenis status yang mungkin dimiliki oleh suatu proses:

- New: status yang dimiliki pada saat proses baru saja dibuat.
- Running: status yang dimiliki pada saat instruksi-instruksi dari sebuah proses dieksekusi.
- Waiting: status yang dimiliki pada saat proses menunggu suatu event (contohnya: proses I/O).
- Ready: status yang dimiliki pada saat proses siap untuk dieksekusi oleh prosesor.
- Terminated: status yang dimiliki pada saat proses telah selesai dieksekusi.

Nama-nama tersebut adalah berdasar opini, istilah tersebut bervariasi di sepanjang sistem operasi. Keadaan yang mereka gambarkan ditemukan pada seluruh sistem. Namun, pada sistem operasi tertentu lebih baik menggambarkan keadaan/status proses. Penting untuk diketahui bahwa hanya satu proses yang dapat berjalan pada prosesor mana pun pada satu waktu. Namun, banyak proses yang dapat berstatus ready atau waiting. Keadaan diagram yang berkaitan dengan keadaan tersebut dijelaskan pada gambar 3.1

Ada tiga kemungkinan bila sebuah proses memiliki status running:



Gambar~3.1: Status proses

- Jika program telah selesai dieksekusi maka status dari proses tersebut akan berubah menjadi Terminated.
- Jika waktu yang disediakan oleh OS untuk proses tersebut sudah habis maka akan terjadi interrupt dan proses tersebut kini berstatus Ready.
- Jika suatu event terjadi pada saat proses dieksekusi (seperti ada request I/O) maka proses tersebut akan menunggu event tersebut selesai dan proses berstatus Waiting.

3.1.3 Proses Control Block

Tiap proses digambarkan dalam sistem operasi oleh sebuah process control block (PCB) - juga disebut sebuah control block. Sebuah PCB ditunjukkan dalam Gambar 3.2. PCB berisikan banyak bagian dari informasi yang berhubungan dengan sebuah proses yang spesi…k, termasuk hal-hal di bawah ini:

- Status proses: status mungkin, new, ready, running, waiting, halted, dan juga banyak lagi.
- Program counter: suatu stack yang berisi alamat dari instruksi selanjutnya untuk dieksekusi untuk proses ini.

- CPU register: Register bervariasi dalam jumlah dan jenis, tergantung pada rancangan komputer. Register tersebut termasuk accumulator, register indeks, stack pointer, general-purposes register, ditambah code information pada kondisi apa pun. Beserta dengan program counter, keadaan/status informasi harus disimpan ketika gangguan terjadi, untuk memungkinkan proses tersebut berjalan/bekerja dengan benar setelahnya (lihat Gambar 3.3Tiap proses digambarkan dalam sistem operasi oleh sebuah process control block (PCB) - juga disebut sebuah control block. Sebuah PCB ditunjukkan dalam Gambar 3-2. PCB berisikan banyak bagian dari informasi yang berhubungan dengan sebuah proses yang spesi…k, termasuk hal-hal di bawah ini:

- Status proses: status mungkin, new, ready, running, waiting, halted, dan juga banyak lagi.

- Program counter: suatu stack yang berisi alamat dari instruksi selanjutnya untuk dieksekusi untuk proses ini.

- CPU register: Register bervariasi dalam jumlah dan jenis, tergantung pada rancangan komputer. Register tersebut termasuk accumulator, register indeks, stack pointer, general-purposes register, ditambah code information pada kondisi apa pun. Beserta dengan program counter, keadaan/status informasi harus disimpan ketika gangguan terjadi, untuk memungkinkan proses tersebut berjalan/bekerja dengan benar setelahnya (lihat Gambar 3.3).

- Informasi managemen memori: Informasi ini dapat termasuk suatu informasi sebagai nilai dari dasar dan batas register, tabel page/halaman, atau tabel segmen tergantung pada sistem memori yang digunakan oleh sistem operasi (lihat Bab 5).

- Informasi pencatatan: Informasi ini termasuk jumlah dari CPU dan waktu riil yang digunakan, batas waktu, jumlah akun jumlah job atau proses, dan banyak lagi.

- Informasi status I/O: Informasi termasuk daftar dari perangkat I/O yang di gunakan pada proses ini, suatu daftar berkas-berkas yang sedang diakses dan banyak lagi.

- PCB hanya berfungsi sebagai tempat penyimpanan informasi yang dapat bervariasi dari proses yang satu dengan yang lain.

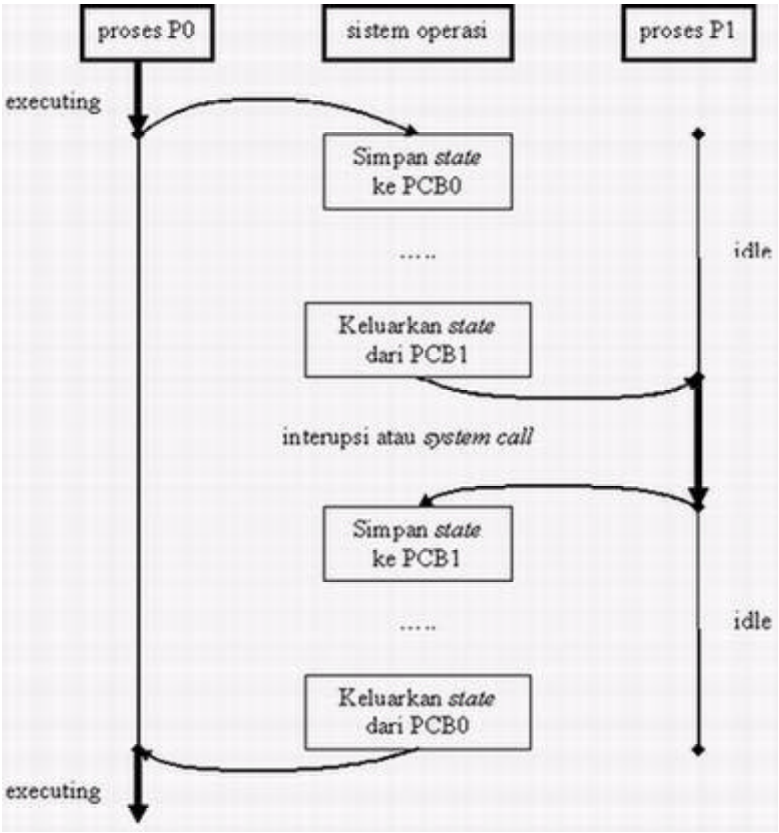
Gambar~3.2: Proses Control Block

3.2 Thread

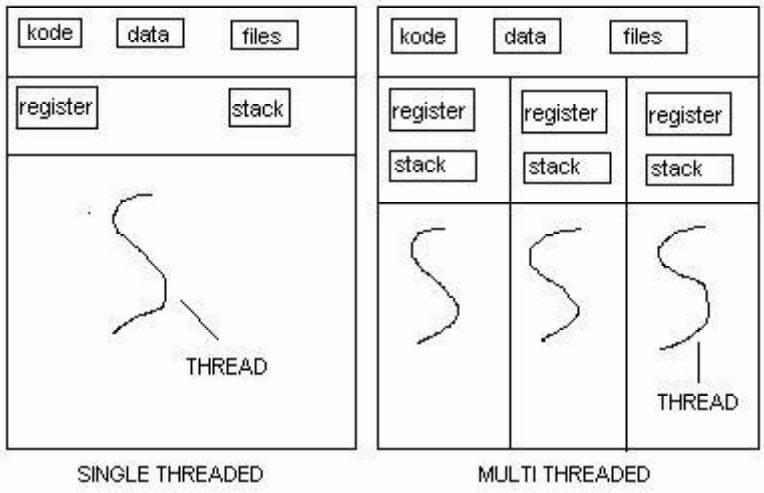
3.2.1 Apa itu thread ?

Thread merupakan unit dasar dari penggunaan CPU, yang terdiri dari Thread_ID, program counter, register set, dan stack. Sebuah thread berbagi code section, data section, dan sumber daya sistem operasi dengan Thread lain yang dimiliki oleh proses yang sama. Thread juga sering disebut lightweight process. Sebuah proses tradisional atau heavyweight process mempunyai thread tunggal yang berfungsi sebagai pengendali. Perbedaan antara proses dengan thread tunggal dengan proses dengan thread yang banyak adalah proses dengan thread yang banyak dapat mengerjakan lebih dari satu tugas pada satu satuan waktu.

Banyak perangkat lunak yang berjalan pada PC modern dirancang secara multi-threading. Sebuah aplikasi biasanya diimplementasi sebagai proses yang terpisah dengan beberapa thread yang berfungsi sebagai pengendali. Contohnya sebuah web browser mempunyai thread untuk menampilkan gam-



Gambar~3.3: Status proses



Gambar~3.4: Thread

bar atau tulisan sedangkan thread yang lain berfungsi sebagai penerima data dari network.

Kadang kala ada situasi dimana sebuah aplikasi diperlukan untuk menjalankan beberapa tugas yang serupa. Sebagai contohnya sebuah web server dapat mempunyai ratusan klien yang mengaksesnya secara concurrent. Kalau web server berjalan sebagai proses yang hanya mempunyai thread tunggal maka ia hanya dapat melayani satu klien pada pada satu satuan waktu. Bila ada klien lain yang ingin mengajukan permintaan maka ia harus menunggu sampai klien sebelumnya selesai dilayani. Solusinya adalah dengan membuat web server menjadi multi-threading. Dengan ini maka sebuah web server akan membuat thread yang akan mendengar permintaan klien, ketika permintaan lain diajukan maka web server akan menciptakan thread lain yang akan melayani permintaan tersebut.

Java mempunyai pengunaan lain dari thread. Perlu diketahui bahwa Java tidak mempunyai konsep asynchronous. Sebagai contohnya kalau program java mencoba untuk melakukan koneksi ke server maka ia akan berada dalam keadaan block state sampai koneksinya jadi (dapat dibayangkan apa yang terjadi apabila servernya mati). Karena Java tidak memiliki konsep asynchronous maka solusinya adalah dengan membuat thread yang mencoba

untuk melakukan koneksi ke server dan thread lain yang pertamanya tidur selamabeberap waktu (misalnya 60 detik) kemudian bangun. Ketika waktu tidurnya habis maka ia akan bangun dan memeriksa apakah thread yang melakukan koneksi ke server masih mencoba untuk melakukan koneksi ke server, kalau thread tersebut masih dalam keadaan mencoba untuk melakukan koneksi ke server maka ia akan melakukan interrupt dan mencegah thread tersebut untuk mencoba melakukan koneksi ke server.

3.2.2 Keuntungan Thread

Keuntungan dari program yang multithreading dapat dipisah menjadi empat kategori:

1. Responsi: Membuat aplikasi yang interaktif menjadi multithreading dapat membuat sebuah program terus berjalan meskipun sebagian dari program tersebut diblok atau melakukan operasi yang panjang, karena itu dapat meningkatkan respons kepada pengguna. Sebagai contohnya dalam web browser yang multithreading, sebuah thread dapat melayani permintaan pengguna sementara thread lain berusaha menampilkan image.

2. Berbagi sumber daya: thread berbagi memori dan sumber daya dengan thread lain yang dimiliki oleh proses yang sama. Keuntungan dari berbagi kode adalah mengizinkan sebuah aplikasi untuk mempunyai beberapa thread yang berbeda dalam lokasi memori yang sama.

3. Ekonomi: dalam pembuatan sebuah proses banyak dibutuhkan pengalokasian memori dan sumber daya. Alternatifnya adalah dengan penggunaan thread, karena thread berbagi memori dan sumber daya proses yang memilikinya maka akan lebih ekonomis untuk membuat dan context switch thread. Akan susah untuk mengukur perbedaan waktu antara proses dan thread dalam hal pembuatan dan pengaturan, tetapi secara umum pembuatan dan pengaturan proses lebih lama dibandingkan thread. Pada Solaris, pembuatan proses lebih lama 30 kali dibandingkan pembuatan thread, dan context switch proses 5 kali lebih lama dibandingkan context switch thread.

4. Utilisasi arsitektur multiprocessor: Keuntungan dari multithreading dapat sangat meningkat pada arsitektur multiprocessor, dimana setiap

thread dapat berjalan secara pararel di atas processor yang berbeda. Pada arsitektur processor tunggal, CPU menjalankan setiap thread secara bergantian tetapi hal ini berlangsung sangat cepat sehingga menciptakan ilusi pararel, tetapi pada kenyataannya hanya satu thread yang dijalankan CPU pada satu-satuan waktu (satu-satuan waktu pada CPU biasa disebut time slice atau quantum).

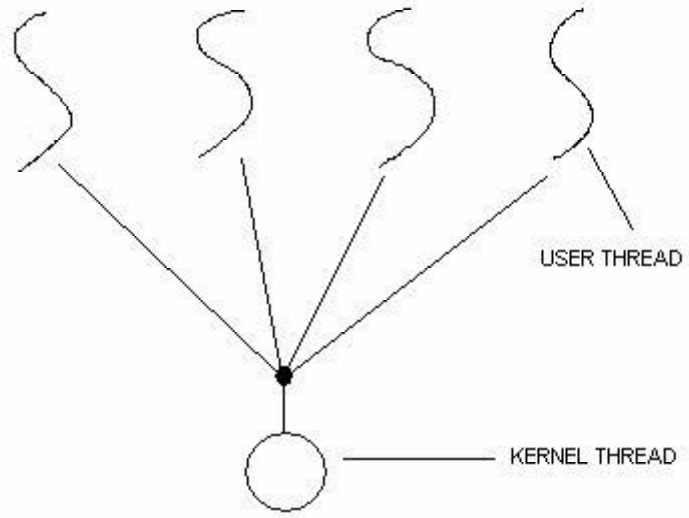
3.2.3 User dan Kernel Thread

User Thread

User thread didukung di atas kernel dan diimplementasi oleh thread library pada user level. Library menyediakan fasilitas untuk pembuatan thread, penjadualan thread, dan managemen thread tanpa dukungan dari kernel. Karena kernel tidak menyadari user-level thread maka semua pembuatan dan penjadualan thread dilakukan di user space tanpa intervensi dari kernel. Oleh karena itu, user-level thread biasanya cepat untuk dibuat dan diatur. Tetapi user thread mempunyai kelemahan yaitu apabila kernelnya merupakan thread tunggal maka apabila salah satu user-level thread menjalankan blocking system call maka akan mengakibatkan seluruh proses diblok walau pun ada thread lain yang dapat jalan dalam aplikasi tersebut. Contoh user-thread libraries adalah POSIX Pthreads, Mach C-threads, dan Solaris threads.

Kernel Thread

Kernel thread didukung langsung oleh sistem operasi. Pembuatan, penjadualan, dan managemen thread dilakukan oleh kernel pada kernel space. Karena pengaturan thread dilakukan oleh sistem operasi maka pembuatan dan pengaturan kernel thread lebih lambat dibandingkan user thread. Keuntungannya adalah thread diatur oleh kernel, karena itu jika sebuah thread menjalankan blocking system call maka kernel dapat menjadualkan thread lain di aplikasi untuk melakukan eksekusi. Keuntungan lainnya adalah pada lingkungan multiprocessor, kernel dapat menjadual thread-thread pada processor yang berbeda. Contoh sistem operasi yang mendukung kernel thread adalah Windows NT, Solaris, Digital UNIX.



Gambar~3.5: Many to one

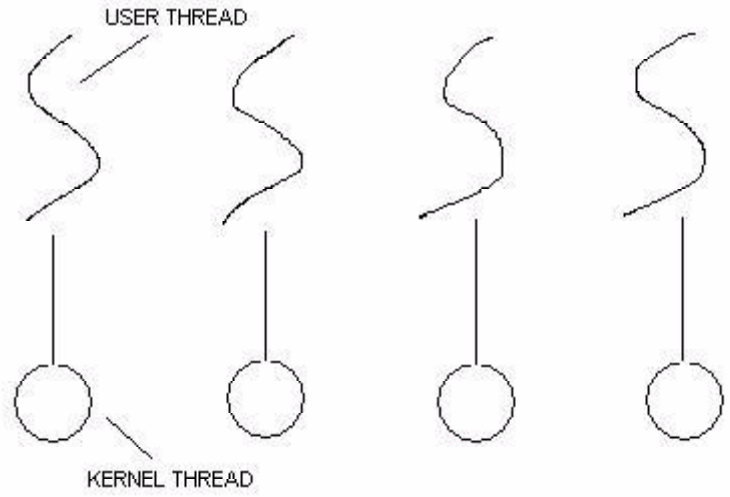
3.2.4 Multithreading Model

Many to one Model

Many-to-One model memetakan banyak user-level thread ke satu kernel thread. Pengaturan thread dilakukan di user space, oleh karena itu ia e...sien tetapi ia mempunyai kelemahan yang sama dengan user thread. Selain itu karena hanya satu thread yang dapat mengakses thread pada suatu waktu maka multiple thread tidak dapat berjalan secara pararel pada multiprocessor. User-level thread yang diimplementasi pada sistem operasi yang tidak mendukung kernel thread menggunakan Many-to-One model.

One to one Model

One-to-One model memetakan setiap user thread ke kernel thread. Ia menyediakan lebih banyak concurrency dibandingkan Many-to-One model. Keuntungannya sama dengan keuntungan kernel thread. Kelemahannya model ini adalah setiap pembuatan user thread membutuhkan pembuatan kernel thread. Karena pembuatan thread dapat menurunkan performa dari sebuah

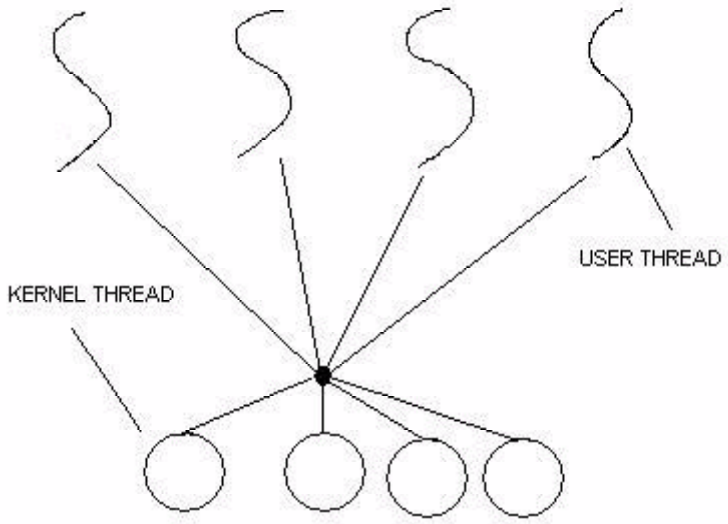


Gambar~3.6: One to one

aplikasi maka implmentasi dari model ini membatasi jumlah thread yang dibatasi oleh sistem. Contoh sistem operasi yang mendukung One-to-One model adalah Windows NT dan OS/2.

Many to many Model

Many-to-many model multiplexes banyak user-level thread ke kernel thread yang jumlahnya lebih kecil atau sama banyaknya dengan user-level thread. Jumlah kernel thread dapat spesi…k untuk sebagian aplikasi atau sebagian mesin. Many-to-One model mengizinkan developer ntuk membuat user thread sebanyak yang ia mau tetapi concurrency tidak dapat diperoleh karena hanya satu thread yang dapat dijadual oleh kernel pada suatu waktu. One-to-One menghasilkan concurrency yang lebih tetapi developer harus hati-hati untuk tidak menciptakan terlalu banyak thread dalam suatu aplikasi (dalam beberapa hal, developer hanya dapat membuat thread dalam jumlah yang terbatas). Many-to-Many model tidak menderita kelemahan dari 2 model di atas. Developer dapat membuat user thread sebanyak yang diperlukan, dan kernel thread yang bersangkutan dapat bejalan secara pararel pada multiprocessor. Dan juga ketika suatu thread menjalankan blocking system



Gambar~3.7: Many to many

call maka kernel dapat menjadualkan thread lain untuk melakukan eksekusi. Contoh sistem operasi yang mendukung model ini adalah Solaris, IRIX, dan Digital UNIX.

3.2.5 Fork dan Exec System Call

Ada dua kemungkinan dalam system UNIX jika fork dipanggil oleh salah satu thread dalam proses:

- Semua thread diduplikasi.
- Hanya thread yang memanggil fork.

Kalau thread memanggil exec System Call maka program yang dispesi-...kasi di parameter exec akan mengganti keseluruhan proses termasuk thread dan LWP.

Penggunaan dua versi dari fork di atas tergantung dari aplikasi. Kalau exec dipanggil seketika sesudah fork, maka duplikasi seluruh thread tidak

dibutuhkan, karena program yang dispesi...kasi di parameter exec akan mengganti seluruh proses. Pada kasus ini cukup hanya mengganti thread yang memanggil fork. Tetapi jika proses yang terpisah tidak memanggil exec sesudah fork maka proses yang terpisah tersebut hendaknya menduplikasi seluruh thread.

3.2.6 Cancellation

Thread cancellation adalah tugas untuk memberhentikan thread sebelum ia menyelesaikan tugasnya. Sebagi contohnya jika dalam program java kita hendak mematikan Java Virtual Machine (JVM) maka sebelum JVM-nya dimatikan maka seluruh thread yang berjalan dihentikan terlebuh dahulu. Thread yang akan diberhentikan biasa disebut target thread.

Pemberhentian target thread dapat terjadi melalui dua cara yang berbeda:

- Asynchronous cancellation: suatu thread seketika itu juga memberhentikan target thread.
- Defered cancellation: target thread secara perodik memeriksa apakah dia harus berhenti, cara ini memperbolehkan target thread untuk memberhentikan dirinya sendiri secara terurut.

Hal yang sulit dari pemberhentian thread ini adalah ketika terjadi situasi dimana sumber daya sudah dialokasikan untuk thread yang akan diberhentikan. Selain itu kesulitan lain adalah ketika thread yang diberhentikan sedang meng-update data yang ia bagi dengan thread lain. Hal ini akan menjadi masalah yang sulit apabila digunakan asynchronous cancellation. Sistem operasi akan mengambil kembali sumber daya dari thread yang diberhentikan tetapi seringkali sistem operasi tidak mengambil kembali semua sumber daya dari thread yang diberhentikan.

Alternatifnya adalah dengan menggunakan de¤ered cancellation. Cara kerja dari de¤ered cancellation adalah dengan menggunakan satu thread yang berfungsi sebagai pengindikasi bahwa target thread hendak diberhentikan. Tetapi pemberhentian hanya akan terjadi jika target thread memeriksa apakah ia harus berhenti atau tidak. Hal ini memperbolehkan thread untuk memeriksa apakah ia harus berhenti pada waktu dimana ia dapat diberhentikan secara aman yang aman. Pthread merujuk tersebut sebagai cancellation points.

Pada umumnya sistem operasi memperbolehkan proses atau thread untuk diberhentikan secara asynchronous. Tetapi Pthread API menyediakan deferred cancellation. Hal ini berarti sistem operasi yang mengimplementasikan Pthread API akan mengizinkan deferred cancellation.

3.2.7 Penanganan Sinyal

Sebuah sinyal digunakan di sistem UNIX untuk notify sebuah proses kalau suatu peristiwa telah terjadi. Sebuah sinyal dapat diterima secara synchronous atau asynchronous tergantung dari sumber dan alasan kenapa peristiwa itu memberi sinyal.

- Semua sinyal (asynchronous dan synchronous) mengikuti pola yang sama:
- Sebuah sinyal dimunculkan oleh kejadian dari suatu persitiwa.
- Sinyal yang dimunculkan tersebut dikirim ke proses.
- Sesudah dikirim, sinyal tersebut harus ditangani.

Contoh dari sinyal synchronous adalah ketika suatu proses melakukan pengaksesan memori secarai ilegal atau pembagian dengan nol, sinyal dimunculkan dan dikirim ke proses yang melakukan operasi tersebut. Contoh dari sinyal asynchronous misalnya kita mengirimkan sinyal untuk mematikan proses dengan keyboard (ALT-F4) maka sinyal asynchronous dikirim ke proses tersebut. Jadi ketika suatu sinyal dimunculkan oleh peristiwa diluar proses yang sedang berjalan maka proses tersebut menerima sinyal tersebut secara asynchronous.

Setiap sinyal dapat ditangani oleh salah satu dari dua penerima sinyal:

- Penerima sinyal yang merupakan set awal dari sistem operasi.
- Penerima sinyal yang dide...nisikan sendiri ole user.

Penanganan sinyal pada program yang hanya memakai thread tunggal cukup mudah yaitu hanya dengan mengirimkan sinyal ke prosesnya. Tetapi mengirimkan sinyal lebih rumit pada program yang multithreading, karena sebuah proses dapat memiliki beberapa thread.

Secara umum ada empat pilihan kemana sinyal harus dikirim:

- Mengirimkan sinyal ke thread yang dituju oleh sinyal tersebut.
- Mengirimkan sinyal ke setiap thread pada proses tersebut.
- Mengirimkan sinyal ke thread tertentu dalam proses.
- Menugaskan thread khusus untuk menerima semua sinyal yang ditujukan pada proses.

Cara untuk mengirimkan sebuah sinyal tergantung dari jenis sinyal yang dimunculkan. Sebagai contoh sinyal synchronous perlu dikirimkan ke thread yang memunculkan sinyal tersebut bukan thread lain pada proses tersebut. Tetapi situasi dengan sinyal asynchronous menjadi tidak jelas. Beberapa sinyal asynchronous seperti sinyal yang berfungsi untuk mematikan proses (contoh: alt-f4) harus dikirim ke semua thread. Beberapa versi UNIX yang multithreading mengizinkan thread menerima sinyal yang akan ia terima dan menolak sinyal yang akan ia tolak. Karena itu sinyal asynchronouns hanya dikirimkan ke thread yang tidak memblok sinyal tersebut. Solaris 2 mengimplementasikan pilihan ke-4 untuk menangani sinyal. Windows 2000 tidak menyediakan fasilitas untuk mendukung sinyal, sebagai gantinya Windows 2000 menggunakan asynchronous procedure calls (APCs). Fasilitas APC memperbolehkan user thread untuk memanggil fungsi tertentu ketika user thread menerima noti…kasi peristiwa tertentu.

3.2.8 Thread Pools

Pada web server yang multithreading ada dua masalah yang timbul:

- Ukuran waktu yang diperlukan untuk menciptakan thread untuk melayani permintaan yang diajukan terlebih pada kenyataannya thread dibuang ketika ia seketika sesudah ia menyelesaikan tugasnya.
- Pembuatan thread yang tidak terbatas jumlahnya dapat menurunkan performa dari sistem.

Solusinya adalah dengan penggunaan Thread Pools, cara kerjanya adalah dengan membuat beberapa thread pada proses startup dan menempatkan mereka ke pools, dimana mereka duduk diam dan menunggu untuk bekerja. Jadi ketika server menerima permintaan maka maka ia akan membangunkan

thread dari pool dan jika thread tersedia maka permintaan tersebut akan dilayani. Ketika thread sudah selesai mengerjakan tugasnya maka ia kembali ke pool dan menunggu pekerjaan lainnya. Bila tidak thread yang tersedia pada saat dibutuhkan maka server menunggu sampai ada satu thread yang bebas.

Keuntungan thread pool:

- Biasanya lebih cepat untuk melayani permintaan dengan thread yang ada dibanding dengan menunggu thread baru dibuat.
- Thread pool membatasi jumlah thread yang ada pada suatu waktu. Hal ini pentingpada sistem yang tidak dapat mendukung banyak thread yang berjalan secara concurrent.

Jumlah thread dalam pool dapat tergantung dari jumlah CPU dalam sistem, jumlah memori …sik, dan jumlah permintaan klien yang concurrent.

Bab 4

Sistem Operasi Terdistribusi

4.1 Apakah sistem operasi terdistribusi ?

Sistem operasi terdistribusi adalah salah satu implementasi dari sistem terdistribusi, di mana sekumpulan komputer dan prosesor yang heterogen terhubung dalam suatu jaringan. Koleksi-koleksi dari objek-objek ini secara tertutup bekerja secara bersama-sama untuk melakukan suatu tugas atau pekerjaan tertentu.

Tujuan utamanya adalah untuk memberikan hasil secara lebih, terutama dalam:

- ...le system
- name space
- waktu pengolahan
- keamanan
- akses ke seluruh resources, seperti prosesor, memori, penyimpanan sekunder, dan perangkat keras.

4.1.1 Sistem Operasi terdistribusi vs Sistem Operasi Jaringan

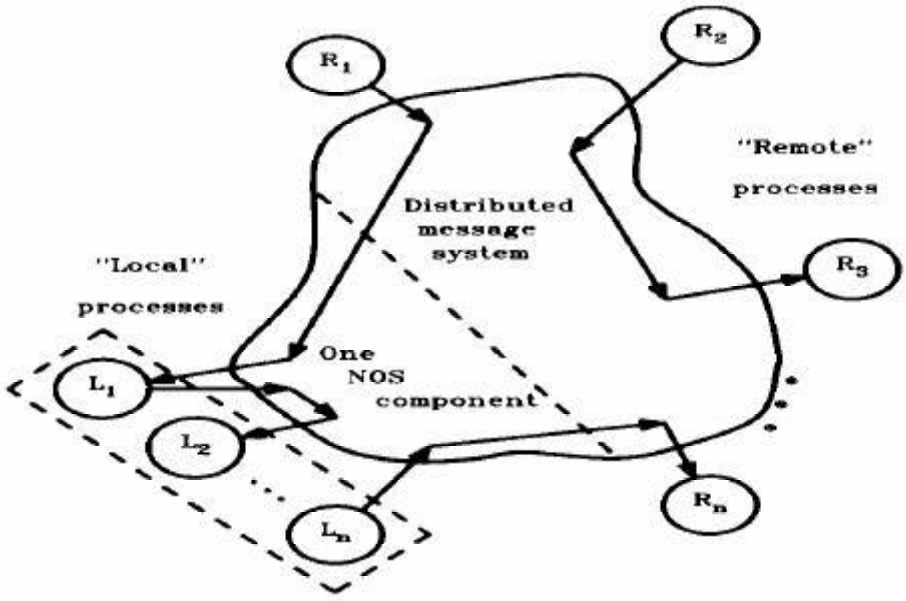
Suatu sistem operasi terdistribusi yang sejati adalah yang berjalan pada beberapa buah mesin, yang tidak melakukan sharing memori, tetapi terlihat bagi user sebagai satu buah komputer single. Pengguna tidak perlu

memikirkan keberadaan perangkat keras yang ada, seperti prosesor. Contoh dari sistem seperti ini adalah Amoeba.

Sistem operasi terdistribusi berbeda dengan sistem operasi jaringan. Untuk dapat membedakannya, sistem operasi jaringan memiliki ciri-ciri sebagai berikut:

- Tiap komputer memiliki sistem operasi sendiri
- Tiap personal komputer memiliki sistem …le sendiri, di mana data-data disimpan
- Sistem operasi tiap komputer dapat berbeda-beda atau heterogen
- Pengguna harus memikirkan keberadaan komputer lain yang terhubung, dan harus mengakses, biasanya menggunakan remote login (telnet)
- File system dapat digunakan dengan dukungan NFS

Contoh dari sistem ini adalah Unix dan Linux Server



Gambar~4.1: Skema Sistem Operasi Jaringan

4.2 Fungsi Sistem Operasi Terdistribusi

Sistem operasi terdistribusi memiliki manfaat dalam banyak sistem dan dunia komputasi yang luas. Manfaat-manfaat ini termasuk dalan sharing resource, waktu komputasi, reliabilitas, dan komunikasi.

4.2.1 Shared Resource

Walaupun perangkat sekarang sudah memiliki kemampuan yang cepat dalam proses-proses komputasi, atau misal dalam mengakses data, tetapi pengguna masih saja menginginkan sistem berjalan dengan lebih cepat. Apabila hardware terbatas, kecepatan yang diinginkan user dapat diatasi dengan menggabung perangkat yang ada dengan sistem DOS (Distributed Operating System).

4.2.2 Manfaat Komputasi

Salah satu keunggulan sistem operasi terdistribusi ini adalah bahwa komputasi berjalan dalam keadaan pararel. Proses komputasi ini dipecah dalam banyak titik (nodes), yang mungkin berupa komputer pribadi, prosesor tersendiri, dan kemungkinan perangkat prosesor-prosesor yang lain. Sistem operasi terdistribusi ini bekerja baik dalam memecah komputasi ini dan baik pula dalam mengambil kembali hasil komputasi dari titik-titik cluster untuk ditampilkan hasilnya.

4.2.3 Reliabilitas

Fitur unik yang dimiliki oleh DOS ini adalah reliabilitas. Berdasarkan design dan implementasi dari design sistem ini, maka hilangnya suatu node tidak akan berdampak terhadap integritas system. Hal ini berbeda dengan komputer personal, apabila ada salah satu hardware yang mengalami kerusakan, maka system akan berjalan tidak seimbang, bahkan sistem bisa tidak dapat berjalan atau mati.

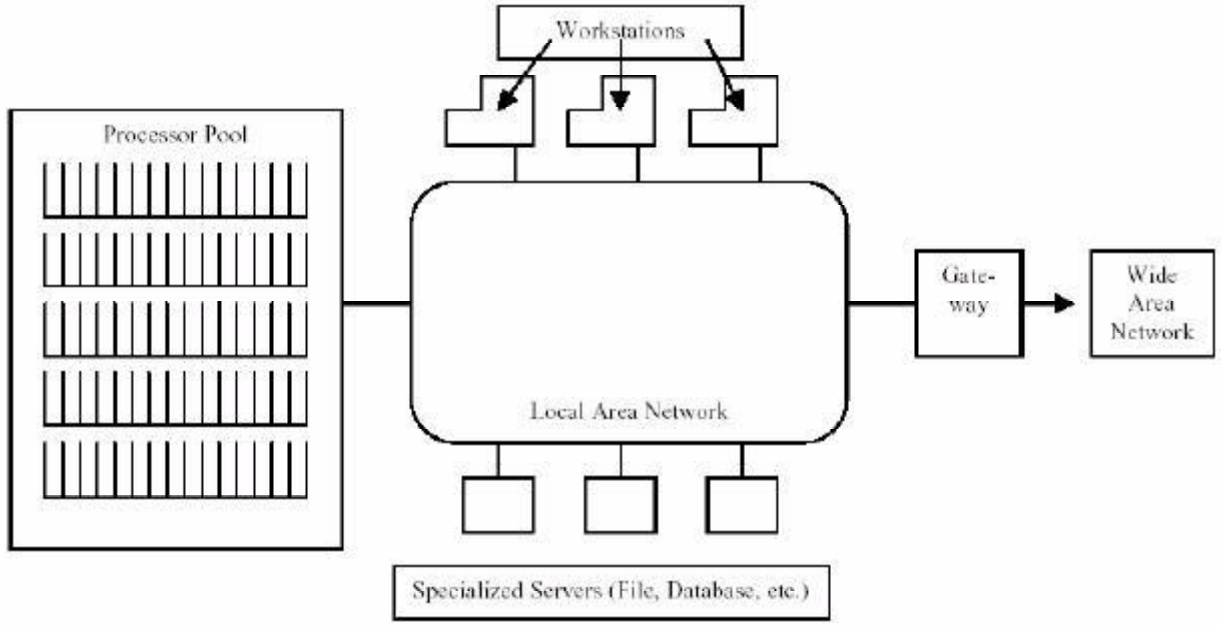
Dalam sistem operasi terdistribusi tadi sebenarnya cara kerjanya mirip dengan personal computer, tetapi bedanya apabila ada node yang mati, maka akan terjadi proses halt terhadap node tersebut dan proses komputasi dapat dialihkan. Hal ini akan membuat sistem DOS selalu memiliki reliabilitas yang tinggi.

4.2.4 Komunikasi

Sistem operasi terdistribusi biasanya berjalan dalam jaringan, dan biasanya melayani koneksi jaringan. Sistem ini biasanya digunakan user untuk proses networking. User dapat saling bertukar data, atau saling berkomunikasi antar titik baik secara LAN maupun WAN.

4.3 Komponen Sistem Operasi

Sistem operasi terdistribusi, yang saat ini akan dibahas sebagai titik tolak adalah Amoeba, yang saat ini banyak digunakan sebagai salah satu implementasi dari sistem operasi terdistribusi itu sendiri. Sistem Amoeba ini tumbuh dari bawah hingga akhirnya tumbuh menjadi sistem operasi terdistribusi.



Design Sistem Operasi Amoeba

Sistem operasi terdistribusi pada umumnya memerlukan hardware secara spesi…k. Komponen utama dalam sistem ini adalah : workstation, LAN, gateway, dan processor pool, seperti yang diilustrasikan pada gambar di atas.

Workstation atau komputer personal mengeksekusi proses yang memerlukan interaksi dari user seperti text editor atau manager berbasis window. Server khusus memiliki fungsi untuk melakukan tugas yang spesi…k. Server

ini mengambil alih proses yang memerlukan I/O yang khusus dari larikan disk. Gateway berfungsi untuk mengambil alih tugas untuk terhubung ke jaringan WAN.

Procesor pool mengambil alih semua proses yang lain. Tiap unit ini biasanya terdiri dari prosesor, memori lokal, dan koneksi jaringan. Tiap prosesor mengerjakan satu buah proses sampai prosesor yang tidak digunakan habis. Untuk selanjutnya proses yang lain berada dalam antrian menunggu proses yang lain selesai. Inilah keunggulan sistem operasi terdistribusi dalam hal reliabilitas. Apabila ada satu unit pemroses yang mati, maka proses yang dialokasikan harus di restart, tetapi integritas sistem tidak akan terganggu, apabila proses deteksi berjalan dengan baik. Desain sistem ini memungkinkan untuk 10 sampai 100 prosesor.

Spesi…kasi perangkat keras yang harus disediakan pada tiap cluster minimalnya adalah :

- File server: 16 MB RAM, 300MB HD, Ethernet card.
- Workstation: 8 MB RAM, monitor, keyboard, mouse
- Pool processor: 4 MB RAM, 3.5 ‡oppy drive

4.3.1 Arsitektur Software

Sistem operasi terdistribusi sejati memiliki arsiitektur software yang unik. Arsitektur software ini dikarakterkan dalam objek di dalam hubungan antara klien dan server. Proses-proses yang terjadi di klien menggunakan remote procedure yang memanggil dan mengirimkan request ke server untuk memproses data atau objek yang dibawa. Tiap objek yang dibawa memiliki karakteristik yang disebut sebagai kapabilitas.

Kapabilitas ini besarnya adalah 128 bits. 48 bits pertama menunjukkan servis mana yang memiliki objek tersebut. 24 bits berikutnya adalah nomor dari objek. 8 bits berikutnya menampilkan operasi yang diijinkan terhadap objek yang bersangkutan. Dan 48 bits terakhir merupakan check …eld yang merupakan …eld yang telah terenkripsi agar tidak dapat dimodi…kasi oleh proses yang lain.

Operasi diselesaikan oleh RPC (remote procedure calls) yang dibuat oleh klien di dalam proses yang kecil dan ringan. Proses dengan tipe seperti

ini memiliki bidang alamat sendiri, dan bisa saja memiliki satu atau lebih hubungan. Hubungan ini ketika berjalan memiliki program counter dan stack sendiri, tetapi dapat saling berbagi kode dan data antara hubungan lain di dalam proses. Ada 3 macam basis panggilan sistem yang dapat digunakan dalam proses yang dimiliki user, yaitu do_operation, get_request, dan send_reply.

Bagian yang pertama mengirimkan pesan ke server, setelah proses memblok sampai server mengirimkan balasan. Server menggunakan panggilan sistem ke dua untuk mengindikasikan bahwa server akan menerima pesan pada port tertentu. Server juga menggunakan panggilan sistem ke tiga untuk mengirimkan kembali informasi ke proses yang dipanggil.

Dengan dibangun dari perintah sistem yang primitif, maka sistem ini menjadi antarmuka untuk program aplikasi. Hal ini diselesaikan oleh tingkat dari pengarahan yang mengijinkan pengguna untuk ber…kir terhadap struktur ini sebagai objek dan operasi-operasi terhadap objek ini.

Berhubungan dengan objek-objek adalah class. Kelas dapat berisi kelas yang lain dan juga hierarki secara alami. Pewarisan membuat antarmuka objek untuk implementasi manipulasi objek seperti menghapus, membaca, menulis, dan sebagainya.

4.3.2 Manajemen Berkas

Dalam sistem operasi terdistribusi ini sistem berkas dipetakan dengan baik dengan berorientasi pada objek yang ada dan kapabilitasnya. Hal ini akan menjadi berkesan abstrak, terutama untuk kelas pengguna. Ada tingkatan yang lebih ekstra dalam pemetaan berkas yang ada, mulai dari simbol, pengurutan nama path, dan kapabilitasnya. Melalui sistem ini objek lokal tidak ada bedanya dengan objek publik.

Dalam sistem ini ada semacan tingkatan akses yang sebenarnya mirip UNIX. Setiap user dan group memiliki hak akses yang berbeda-beda pada setiap berkas atau folder yang ada pada sistem operasi terdistribusi.

Dalam implementasi sistem Amoeba, terutama di negeri Belanda, hak akses yang dimiliki pengguna terbatas pada hak baca …le, tulis/membuat …le, dan hapus …le. Dengan hal ini, maka keamanan server dapat terjaga.

Pelayanan terhadap direktori yang ada dibuat sangat ketat dalam hal keamanan. Bahkan dibuat semacan kode acak yang akan menyandikan …le

tersebut sehingga tidak mudah dibaca oleh siapapun. Kode penyandinya akan digunakan lagi oleh sistem untuk mengembalikan …le seperti semula kepada user.

Kode ini hanya akan diberikan kepada pemilik …le tersebut. Jadi ketika user mengakses …le/berkas yang bersangkutan, maka kode penyandi akan dibuat oleh sistem, agar pemilik …le dapat membacanya.

Pelayanan direktori ini juga bertanggungjawab dalam hal backup sistem. Hal ini akan menyebabkan …le selalu berada dalam keadaan yang aman, dan lebih kebal tehadap gangguan yang terjadi di dalam sistem, karena pelayanan direktori ini menyimpan cache dari …le atau direktori yang berada pada sistem.

4.4 Proses

Dalam sistem operasi terdistribusi yang sejati, tiap proses berada pada alamat segmen-segmen virtual. Proses-proses ini dapat memiliki lebih dari satu hubungan. Kaitan-kaitan ini dialokasikan ke prosesor-prosesor sampai semua prosesor habis digunakan. Hasil dari manajemen proses seperti ini menghasilkan utilisasi yang lebih baik, di mana tidak perlu switch apabila harus ada proses yang berat, karena satu proses dialokasikan ke satu prosesor. Sedangkan untuk proses yang tidak kebagian tempat, maka akan masuk ke antrian. Kaitan-kaitan proses ini menggunakan semaphore untuk menunjukkan akti…tasnya

Masing- masing proses memiliki kontrol sendiri pada spasi alamatnya. Masing-masing proses dapat menambah atau menghapus segmen dari spasi alamat virtualnya melalui operasi pemetaan. Objek seperti …le yang berisi kapabilitas, dan yang membaca adalah kernel, dan apabila proses diijinkan, maka ia dapat memetakan atau menghapus pemetaan segmen pada alamat virtualnya.

Untuk membangun sebuah proses, maka pendekripsi proses mengirimkannya ke kernel. Hal ini diketahui sebagai pengiriman request untuk proses. Sebuah deskriptor proses dapat berisi deskriptor host, kapabilitas proses, penanganan kapabilitas, dan juga jumlah segmen. Deskriptor host berisi proses ini memiliki jenis apa, dan dapat berjalan di mana. Isinya adalah baris instruksi, kebutuhan memori, kelas mesin, informasi, dan sebagainya. Kernel harus memiliki deskriptor host yang sama untuk melanjutkan proses.

Kapabilitas proses adalah memiliki tingkatan lebih tinggi dari proses, yang mengatur apa yang dapat dilakukan oleh proses, atau proses ini hanya dapat dilakukan oleh siapa. Pengatur kapabilitas mirip dengan hal ini, tetapi hanya bekerja untuk proses yang tidak normal.

Alamat proses terenkapsulasi di dalam peta memori internal. Peta ini meiliki entri untuk setiap segmen dari alamat untuk proses yang potensial. Entri berisi alamat virtual, panjang segmen, pemetaan segmen, dan kapabilitas dari objek yang mengetahui dari mana objek tersebut diinisialisasi..

Ada juga kaitan pemetaan yang mendeskripsikan atribut yang lain, termasuk di antaranya mende…nisikan inisial keadaan dari kaitan, status prosesor, program counter, stack pointer, stack base, nilai register, dan keadaan sistem pemanggil. Hal ini mengijinkan deskriptor untuk digunakan di proses.

Proses memiliki dua macam keadaan, yaitu proses sedang berjalan atau sedang stunned. Stunned terjadi bila proses masih ada, tetapi tidak melakukan eksekusi apapun, atau sedang dalam proses debug. Pada keadaan ini kernel memberitahu komunikator (kernel yang lain) adanya proses yang dalam keadaan stunned. Kernel yang lain tersebut berusaha berkomunikasi dengan proses itu sampai proses di-kill atau proses tersebut berjalan kembali. Debugging dan migrasi pada proses ini selesai setelah adanya stunning.

Bab 5

File Service

5.1 Pengenalan

Presently, our most common exposure to distributed systems that exemplify some degree of transparency is through distributed …le systems. We'd like remote …les to look and feel just like local ones.

A …le system is responsible for the organization, storage, retrieval, naming, sharing, and protection of …les. File systems provide directory services, which convert a …le name (possibly a hierarchical one) into an internal identi…er (e.g. inode, FAT index). They contain a representation of the …le data itself and methods for accessing it (read/write). The …le system is responsible for controlling access to the data and for performing low-level operations such as bu¤ering frequentlyused data and issuing disk I/O requests.

Our goals in designing a distributed …le system are to present certain degrees of transparency to the user and the system:

- access transparency

  Clients are unaware that …les are distributed and can access them in the same way as local …les are accessed.

- location transparency

  A consistent name space exists encompassing local as well as remote …les. The name of a …le does not give it location.

- concurrency transparency

All clients have the same view of the state of the …le system. This means that if one process is modifying a …le, any other processes on the same system or remote systems that are accessing the …les will see the modi…cations in a coherent manner.

- failure transparency

  The client and client programs should operate correctly after a server failure.

- heterogeneity

  File service should be provided across di¤erent hardware and operating system platforms.

- scalability

  The …le system should work well in small environments (1 machine, a dozen machines) and also scale gracefully to huge ones (hundreds through tens of thousands of systems).

- replication transparency

  To support scalability, we may wish to replicate …les across multiple servers. Clients should be unaware of this.

- migration transparency

  Files should be able to move around without the client's knowledge.

5.1.1 Konsep Sistem Files terdistribusi

A …le service is a speci…cation of what the …le system o¤ers to clients. A …le server is the implementation of a …le service and runs on one or more machines.

A …le itself contains a name, data, and attributes (such as owner, size, creation time, access rights). An immutable …le is one that, once created, cannot be changed. Immutable …les are easy to cache and to replicate across servers since their contents are guaranteed to remain unchanged.

Two forms of protection are generally used in distributed …le systems, and they are essentially the same techniques that are used in single-processor non-networked systems:

- capabilities

  Each user is granted a ticket (capability) from some trusted source for each object to which it has access. The capability speci…es what kinds of access are allowed.

- access control lists

  Each …le has a list of users associated with it and access permissions per user. Multiple users may be organized into an entity known as a group.

5.1.2 Jenis File Service

To provide a remote system with …le service, we will have to select one of two models of operation. One of these is the upload/download model. In this model, there are two fundamental operations: read …le transfers an entire …le from the server to the requesting client, and write …le copies the …le back to the server. It is a simple model and e¢cient in that it provides local access to the …le when it is being used. Three problems are evident. It can be wasteful if the client needs access to only a small amount of the …le data. It can be problematic if the client doesn't have enough space to cache the entire …le. Finally, what happens if others need to modify the same …le? The second model is a remote access model. The …le service provides remote operations such as open, close, read bytes, write bytes, get attributes, etc. The …le system itself runs on servers. The drawback in this approach is the servers are accessed for the duration of …le access rather than once to download the …le and again to upload it.

Another important distinction in providing …le service is that of understanding the di¤erence between directory service and …le service. A directory service, in the context of …le systems, maps human-friendly textual names for …les to their internal locations, which can be used by the …le service. The …le service itself provides the …le interface (this is mentioned above). Another component of …le distributed …le systems is the client module. This is the client-side interface for …le and directory service. It provides a local …le system interface to client software (for example, the vnode …le system layer of a UNIX kernel).

5.2 Komponen File Service

5.2.1 Naming

In designing a distributed …le service, we should consider whether all machines (and processes) should have the exact same view of the directory hierarchy. We might also wish to consider whether the name space on all machines should have a global root directory (a.k.a. super root) so that …les can be accessed as, for example, //server/path. This is a model that was adopted by the Apollo Domain System, an early distributed …le system, and more recently by the web community in the construction of a uniform resource locator (URL).

In considering our goals in name resolution, we must distinguish between location transparency and location independence. By location transparency we mean that the path name of a …le gives no hint to where the …le is located. For instance, we may refer to a …le as //server1/dir/…le. The server (server) can move anywhere without the client caring, so we have location transparency. However, if the …le moves to server2 things will not work. If we have location independence, the …les can be moved without their names changing. Hence, if machine or server names are embedded into path names we do not achieve location independence.

It is desirable to have access transparency, so that applications and users can access remote …les just as they access local …les. To facilitate this, the remote …le system name space should be syntactically consistent with the local name space. One way of accomplishing this is by rede…ning the way …les are named and require an explicit syntax for identifying remote …les. This can cause legacy applications to fail and user discontent (users will have to learn a new way of naming their …les). An alternate solution is to use a …le system mounting mechanism to overlay portions of another …le system over a node in a local directory structure. Mounting is used in the local environment to construct a uniform name space from separate …le systems (which reside on di¤erent disks or partitions) as well as incorporating special-purpose …le systems into the name space (e.g. /proc on many UNIX systems allows …le system access to processes). A remote …le system can be mounted at a particular point in the local directory tree. Attempts to access …les and directories under that node will be directed to the driver for that …le system.

To summarize, our naming options are:

- machine and path naming (machine:path, ./machine/path).
- mount remote …le systems onto the local directory hierarchy (merging the two name spaces).
- provide a single name space which looks the same on all machines.

The …rst two of these options are relatively easy to implement.

Tipe Nama

When we talk about …le names, we refer to symbolic names (for example, server.c). These names are used by people (users or programmers) to refer to …les. Another name is the identi…er used by the system internally to refer to a …le. We can think of this as a binary name (more precisely, as an address). On most UNIX …le systems, this would be the device number and inode number. On MS-DOS systems, this would be the drive letter and FAT index.

Directories provide a mapping from symbolic names to …le addresses (binary names). Typically, one symbolic name maps to one …le address. If multiple symbolic names map onto one binary name, these are called hard links. On inode-based …le systems (e.g., most UNIX systems), hard links must exist within the same device since the address (inode) is unique only on that device. On MS-DOS systems, they are not supported because …le attributes are stored with the name of the …le. Having two symbolic names refer to the same data will cause problems in synchronizing …le attributes (how would you locate other …les that point to this data?). A hack to allow multiple names to refer to the same …le (whether its on the same device or a di¤erent device) is to have the symbolic name refer to a single …le address but that …le may have an attribute to tell the system that its contents contain a symbolic …le name that should be dereferenced. Essentially, this adds a level of indirection: access a …le which contains another …le name, which references the …le attributes and data. These …les are known as symbolic links. Finally, it is possible for one symbolic name to refer to multiple …le addresses. This doesn't make much sense on a local system1, but can be useful on a networked …le system to provide fault tolerance or enable the system to use the …le address which is most e¢cient.

5.2.2 File Sharing Semantik

The analysis of …le sharing semantics is that of understanding how …les behave. For instance, on most systems, if a read follows a write, the read of that location will return the values just written. If two writes occur in succession, the following read will return the results of the last write. File systems that behave this way are said to observe sequential semantics.

Sequential semantics can be achieved in a distributed system if there is only one server and clients do not cache data. This can cause performance problems since clients will be going to the server for every …le operation (such as single-byte reads). The performance problems can be alleviated with client caching. However, now if the client modi…es its cache and another client reads data from the server, it will get obsolete data. Sequential semantics no longer hold.

One solution is to make all the writes write-through to the server. This is ine¢cient and does not solve the problem of clients having invalid copies in their cache. To solve this, the server would have to notify all clients holding copies of the data.

Another solution is to relax the semantics. We will simply tell the users that things do not work the same way on the distributed …le system as they did on the local …le system. The new rule can be changes to an open …le are initially visible only to the process (or machine) that modi…ed it. These are known as session semantics.

Yet another solution is to make all the …les immutable2. That is, a …le cannot be open for modi…cation, only for reading or creating. If we need to modify a …le, we'll create a completely new …le under the old name. Immutable …les are an aid to replication but they do not help with changes to the …le's contents (or, more precisely, that the old …le is obsolete because a new one with modi…ed contents succeeded it). We still have to contend with the issue that there may be another process reading the old …le. It's possible to detect that a …le has changed and start failing requests from other processes.

A …nal alternative is to use atomic transactions. To access a …le or a group of …les, a process …rst executes a begin transaction primitive to signal that all future operations will be executed indivisibly. When the work is completed, an end transaction primitive is executed. If two or more transactions start

at the same time, the system ensures that the end result is as if they were run in some sequential order. All changes have an all or nothing property.

5.2.3 Chaching

We can employ caching to improve system performance. There are four places in a distributed system where we can hold data:

1. on the server's disk
2. in a cache in the server's memory
3. in the client's memory
4. on the client's disk

The …rst two places are not an issue since any interface to the server can check the centralized cache. It is in the last two places that problems arise and we have to consider the issue of cache consistency. Several approaches may be taken:

- write-through

  What if another client reads its own cached copy? All accesses would require checking with the server …rst (adds network congestion) or require the server to maintain state on who has what …les cached. Write-through also does not alleviate congestion on writes.

- delayed writes

  Data can be bu¤ered locally (where consistency su¤ers) but …les can be updated periodically. A single bulk write is far more e¢cient than lots of little writes every time any …le contents are modi…ed. Unfortunately the semantics become ambiguous.

- write on close

  This is admitting that the …le system uses session semantics.

- centralized control

  Server keeps track of who has what open in which mode. We would have to support a stateful system and deal with signaling tra¢c.

Bab 6

Name Service

6.1 Pengenalan

Pengaksesan resource pd sistem terdistribusi memerlukan:

- Nama resource (untuk pemanggilan).
- Alamat (lokasi resource tsb).
- Rute (bagaimana mencapai lokasi tsb).

  Konsentrasi pada aspek penamaan, dan pemetaan antara nama & alamat, bukan pada masalah rute, yg dibahas di Jaringan Komputer.

Yang dimaksud dengan resource adalah : komputer, layanan, remote object, berkas, pemakai.

Berikut contoh naming pd aplikasi sistem terdistribusi:

- URL utk mengakses suatu halaman web.
- Alamat e-mail utk komunikasi antar pemakai.

Naming sering dianggap remeh, tapi mendasar dlm sistem terdistribusi. Karena dalam hal ini name berfungsi sebagai identi…er (pengenal) pada sistem

6.1.1 Tujuan Penamaan

- Identi…kasi:

  Seorang pemakai menginginkan obyek/layanan A, bukan obyek/layanan B.

- Memungkinkan terjadinya sharing

  Lebih dari satu pemakai dapat mengindenti…kasikan resource dengan nama yang sesuai (tidak harus nama yang sama).

- Memungkinkan location independence:

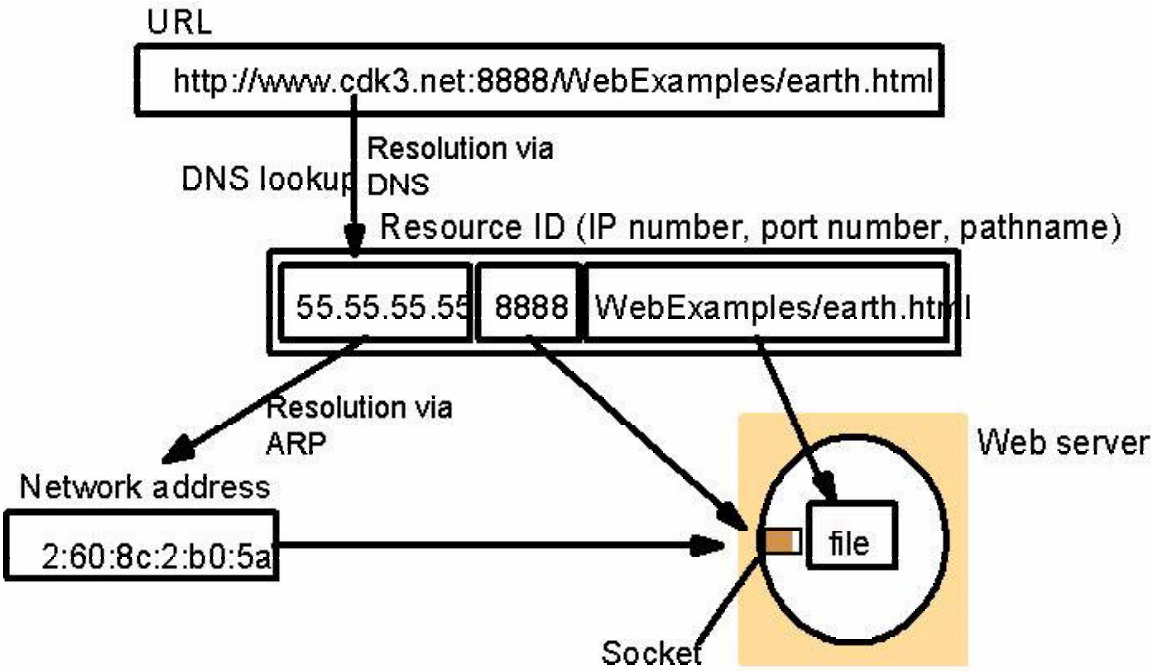
  Perubahan lokasi tidak menuntut perubahan nama, asalkan lokasi tidak menjadi bagian dari nama resource tsb.

- Memberikan kemampuan keamanan (security)

  - Jika sebuah nama dipilih secara acak dari himpunan besar interger, maka nama tsb hanya bisa diketahui dari legitimate source, bukan dari menebak.
  - Jadi jika seseorang mengetahui nama obyek tsb, maka dia memang diberitahu, karena sulit sekali menebak nama tsb.

6.1.2 Contoh Penamaan yang memberikan kemampuan keamanan

Nama dipilih secara acak dari 128 bit integer -> ada sekitar 3 x 1038 nama yang berbeda. Jika sekumpulan obyek membutuhkan nama yang unik, dan di-generate 1 juta dalam 1 detik selama 100 tahun, maka pada akhirnya akan ada sekitar 3 x 1015 obyek (nama). Proporsi nama yang dipakai, jauh lebih kecil dari keseluruhan nama yang tersedia. Probabilitas benar dalam menebak nama obyek tsb adalah 1:1023. Jika dalam dalam 1 detik dilakukan 1 juta tebakan, maka diperlukan sekitar 1010 tahun untuk menebak nama yang benar.



Ilustrasi kerja name service

6.1.3 Jenis Nama

User names:

- Dibuat oleh pemakai (user).
- Merujuk pada suatu obyek atau layanan.
- Terdiri dari strings of characters.

Contoh: hp201 untuk pencetak, ~bettyp/tmp/test.c untuk berkas.

System names:

- Terdiri dari bit string.
- Internal untuk sistem, tidak ditujukan untuk manusia.
- Lebih compact dari user names, shg dapat dibandingkan dengan lebih e...sien.

6.1.4 Struktur Nama

Primitive/‡at names (Unique Identi…ers = UIDs)

- ² Tanpa struktur internal, hanya string of bits.
- ² Digunakan utk perbandingan dengan UID lain.
- ² Tidak membawa informasi lain -> pure names.
- ² Sangat berguna & banyak digunakan karena:
  - Location & application independent, shg tidak menjadi masalah bagi mobilitas obyek.
  - Seragam, …xed size.
  - Compact: mudah disimpan, di-pass, & jika cukup besar menjadi sulit ditebak.

Partitioned Names (PN)

- ² Komposisi dari beberapa nama primitif, biasanya disusun secara hirarkis.

  Contoh: www.gunadarma.ac.id/cs/docs/akademik/SisDis/naming.ppt.
- ² Membawa informasi -> impure names.
- ² Biasanya tidak secara unik mengidenti…kasikan obyek, beberapa nama bisa dipetakan ke satu obyek (e.g. UNIX …le links).

Descriptive names (DN)

- ² Daftar atribut yang secara bersama-sama mengidenti…kasikan obyek secara unik.
- ² Membawa informasi -> impure names.
- ² DN adalah superset dari PN.

6.1.5 Tujuan Fasilitas Penamaan

- E...sien, karena fasilitas penamaan merupakan dasar pada sisdis & digunakan secara terus menerus.
- Terdistribusi. Renungkan jika UIDs dibangkitkan oleh centralized generator.
  - Bottleneck.
  - Node tempat generator tsb mengalami kegagalan.
- Tampak seperti global space, tidak tergantung konekti...tas, topologi, dan lokasi obyek.
- Mendukung pemetaan 1:many antara nama & obyek, untuk memungkinkan multicast.
- Mendukung dynamic relocation of objects, jika obyek/proses potensial untuk mobile (berpindah-pindah). Jadi diperlukan dynamic binding antara nama & alamat, juga antara alamat & rute.
- Memungkinkan local aliases, shg pemakai dapat mengekspresikan interpretasi semantik mereka thdp suatu obyek. Tentu saja diperlukan pemetaan antara aliases dan full names.